An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem

AUTHOR(S):

Imamichi, Takashi; Yagiura, Mutsunori; Nagamochi, Hiroshi

An iterated local search algorithm based on nonlinear programming for the irregular strip packing problem

Takashi Imamichi [a,*], Mutsunori Yagiura [b], Hiroshi Nagamochi [a]

[a] *Department of Applied Mathematics and Physics, Graduate School of Informatics, Kyoto University, Yoshida hommachi, Sakyo-ku, Kyoto 606-8501, Japan*

[b] *Department of Computer Science and Mathematical Informatics, Graduate School of Information Science, Nagoya University, Furocho, Chikusa-ku, Nagoya 464-8603, Japan*

Abstract

The irregular strip packing problem is a combinatorial optimization problem that requires to place a given set of 2-dimensional polygons within a rectangular container so that no polygon overlaps with other polygons or protrudes from the container, where each polygon is not necessarily convex. The container has a fixed width, while its length can change so that all polygons are placed in it. The objective is to find a layout of the set of polygons that minimizes the length of the container.

We propose an algorithm that separates overlapping polygons based on nonlinear programming, and an algorithm that swaps two polygons in a layout so as to find their new positions in the layout with the least overlap. We incorporate these algorithms as components into an iterated local search algorithm for the overlap minimization problem and then develop an algorithm for the irregular strip packing problem using the iterated local search algorithm. Computational comparisons on representative instances disclose that our algorithm is competitive with other existing algorithms. Moreover, our algorithm updates several best known results.

Key words: irregular strip packing problem, iterated local search, no-fit polygon, unconstrained nonlinear programming

* Corresponding author. Tel.: +81-75-753-4921; fax: +81-75-753-4920.
Email addresses: ima@amp.i.kyoto-u.ac.jp (Takashi Imamichi), yagiura@nagoya-u.jp (Mutsunori Yagiura), nag@amp.i.kyoto-u.ac.jp (Hiroshi Nagamochi).

1 Introduction

The *irregular strip packing problem* is a combinatorial optimization problem that requires to place a given set of 2-dimensional polygons within a rectangular container, where each polygon is not necessarily convex, so that no polygon overlaps with other polygons or protrudes from the container. We say that such a layout is *feasible*. The container has a fixed width, while its length can change so that all polygons are placed in it. The objective is to find a feasible layout that minimizes the length of the container. The irregular strip packing problem has a few variations depending on rotations of polygons: (1) rotations of any angle are allowed, (2) a finite number of angles are allowed, (3) no rotation is allowed. Among them, we deal with case (2) in this paper. Note that case (3) is a special case of (2) in which the number of given orientations for each polygon is one. The irregular strip packing problem has many applications in material industry such as paper and textile industries, where raw materials are usually given in rolls. In textile industry, rotations are usually restricted to 180 degrees because textiles have the grain and may have a drawing pattern. The irregular strip packing problem is known to be NP-hard even without rotation [1].

Adamowicz and Albano [2] proposed an algorithm that partitions a given set of polygons into several subsets of polygons, then generates for each of the subsets a rectangle enclosure in which the polygons in the subset are placed compactly (i.e., being with a little wasted space), and finally finds a layout of these enclosures. Albano and Sapuppo [3] gave an algorithm that places given polygons one by one at the bottom-left position according to a sequence of the input polygons, where they used tree search to obtain a good sequence. Some approaches for finding a good sequence are based on local search [4,5].

Mathematical programming was also used for the irregular strip packing problem. *Compaction* and *separation* algorithms based on linear programming have been proposed, e.g., by Li and Milenkovic [6], Bennell and Dowsland [7], and Gomes and Oliveira [8]. Given a feasible layout of given polygons in a container, a *compaction* algorithm translates the polygons in the current layout continuously in order to minimize the length of the container, and it outputs a locally optimal solution. Given an infeasible layout of the given polygons, a *separation* algorithm translates the polygons in the current layout continuously in order to remove the overlap of the polygons.

Bennell and Dowsland [7] combined the bottom-left method and the linear programming based *compaction* algorithm to obtain an algorithm with better performance. Gomes and Oliveira [8] hybridised the bottom-left heuristics and the linear programming based *compaction* and *separation* algorithms. They further incorporated the method into simulated annealing, and the resulting algorithm updated many best known results on the benchmark instances of the irregular strip packing

problem. Burke et al. [9] developed a bottom-left fill heuristic algorithm, and utilized it with hill climbing or tabu search to obtain high quality solutions quickly. Egeblad et al. [10] developed an efficient method that finds a best position of a specified polygon that minimizes its overlap area with the current layout by translating the polygon in a specified direction. and they utilized it in guided local search. See a review by Hopper and Turton [11] for more on the strip packing problem including the irregular strip packing problem.

In this paper, we propose a new *separation* algorithm based on nonlinear programming. We also give an algorithm that swaps two specified polygons in a layout of polygons so that the overlap in the layout is minimized provided that the positions of the other polygons in a given layout are fixed. We incorporate these algorithms as components into an iterated local search algorithm whose objective is to minimize the total amount of overlap and protrusion of a layout, where a layout may not be completely contained in the container during the algorithm. We then develop an algorithm for the irregular strip packing problem using the iterated local search algorithm, which we call ILSQN. Computational comparisons on representative benchmark instances disclose that our algorithm is competitive with other existing algorithms. Moreover, our algorithm updates several best known results.

This paper is organized as follows. We formulate the irregular strip packing problem and illustrate our approach in Section 2. We then define functions that measure the amount of overlap and show how to evaluate these functions and their gradients in Section 3. We propose an iterated local search algorithm for the overlap minimization problem in Section 4 and describe two procedures used in the iterated local search algorithm: a *separation* algorithm based on nonlinear programming and an operation of swapping two polygons in Section 5 and Section 6, respectively. Finally we show computational results in Section 7 and make a concluding remark in Section 8.

2 Formulation and Approach

This section gives a mathematical formulation of the irregular strip packing problem and then illustrates an overview of our approach to the problem. For the irregular strip packing problem, we are given a list $\mathcal{P} = (P_1, \ldots, P_n)$ of polygons, a list $\mathcal{O} = O_1 \times \cdots \times O_n$ of the polygons' orientations, where O_i $(1 \leq i \leq n)$ denotes a set of orientations in which P_i can be rotated, and a rectangular container $C = C(W, L)$ with a width $W \geq 0$ and a length L, where W is a constant and L is a nonnegative variable. Polygons in $\mathcal{P}$ may not be convex.

We denote polygon $P_i \in \mathcal{P}$ rotated in degree $o \in O_i$ by $P_i(o)$, which may be written as P_i for simplicity when the orientation is not specified or clear from the context. For convenience, we regard each of polygons $P_i(o)$ $(i = 1, \ldots, n)$ and

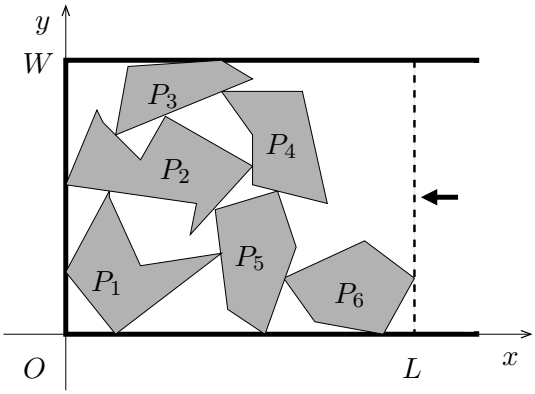

Fig. 1. An example of a feasible layout of six polygons in container $C(W, L)$ (O is the origin).

rectangle C as the set of points inside it including the points on the boundary. For a polygon S, let $\mathrm{int}(S)$ be the interior of S, ∂S be the boundary of S, $\overline{S}$ be the complement of S, and $\mathrm{cl}(S)$ be the closure of S. We describe translations of polygons by Minkowski sums as follows. Let $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ $(i = 1, \ldots, n)$ be a translation vector for P_i. Thus the polygon obtained by translating polygon P_i by $\boldsymbol{x}_i$ is $P_i \oplus \boldsymbol{x}_i = \{\boldsymbol{p} + \boldsymbol{x}_i \mid \boldsymbol{p} \in P_i\}$. See Section 13.3 in [12] for more details on Minkowski sums. Recall that $L \geq 0$ is the length of the container C, which is a decision variable to be minimized. The irregular strip packing problem is formally described as follows:

$$
\begin{array}{ll}
\text{minimize} & L \\
\text{subject to} & \mathrm{int}(P_i(o_i) \oplus \boldsymbol{x}_i) \cap (P_j(o_j) \oplus \boldsymbol{x}_j) = \emptyset, \quad 1 \leq i < j \leq n, \\
& (P_i(o_i) \oplus \boldsymbol{x}_i) \subseteq C(W, L), \quad 1 \leq i \leq n, \\
& L \in \mathbb{R}_+, \\
& o_i \in O_i, \quad 1 \leq i \leq n, \\
& \boldsymbol{x}_i \in \mathbb{R}^2, \quad 1 \leq i \leq n.
\end{array}
\tag{1}
$$

We represent a solution to problem (1) with a pair of n-tuples $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and $\boldsymbol{o} = (o_1, \ldots, o_n)$. Note that a solution $(\boldsymbol{x}, \boldsymbol{o})$ uniquely determines the layout of the polygons. The minimum length L of the container C is formally defined by function

$$
\begin{aligned}
\mu(\boldsymbol{x}, \boldsymbol{o}) = {} & \max\{x_1 \mid (x_1, x_2) \in P_i(o_i) \oplus \boldsymbol{x}_i,\ P_i \in \mathcal{P}\} \\
& - \min\{x_1 \mid (x_1, x_2) \in P_i(o_i) \oplus \boldsymbol{x}_i,\ P_i \in \mathcal{P}\}.
\end{aligned}
\tag{2}
$$

Figure 1 shows an example of a feasible layout of polygons. The length L is decided as described by (2).

2.1 *Overlap Minimization Problem*

The problem (1) contains three types of variables, L, $\boldsymbol{x}$ and $\boldsymbol{o}$. To construct a building block of an entire algorithm to problem (1), we first introduce the *overlap minimization problem*, which requires to find a feasible solution in container C with a fixed length L. For this purpose, we allow polygons to overlap each other and/or protrude from the container during construction of solutions; the amount of overlap and protrusion is penalized in such a way that any solution with no penalty corresponds to a feasible layout to the current container. More specifically, for a pair $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and $\boldsymbol{o} = (o_1, \ldots, o_n)$ of lists of translation vectors and orientations of all polygons, let $f_{ij}(\boldsymbol{x}, \boldsymbol{o})$ be a function that measures the amount of overlap of $P_i(o_i)$ and $P_j(o_j)$, and $g_i(\boldsymbol{x}, \boldsymbol{o})$ be a function that measures the amount of protrusion of $P_i(o_i)$ from the container. Then the overlap minimization problem is formulated by

$$\begin{aligned} &\text{minimize} \quad F(\boldsymbol{x}, \boldsymbol{o}) = \sum_{1 \le i < j \le n} f_{ij}(\boldsymbol{x}, \boldsymbol{o}) + \sum_{1 \le i \le n} g_i(\boldsymbol{x}, \boldsymbol{o}) \\ &\text{subject to} \quad \boldsymbol{x} \in \mathbb{R}^{2n}, \quad \boldsymbol{o} \in \mathcal{O}. \end{aligned} \tag{3}$$

To solve the overlap minimization problem (3), an effective procedure to the problem will be the heart of our algorithm.

Note that the problem (3) is an unconstrained nonlinear programming problem. However, we do not attempt to solve this problem by applying a nonlinear programming method directly since the variables $\boldsymbol{o}$ for rotations are discrete. We treat only variables $\boldsymbol{x}$ for translations while fixing the variables $\boldsymbol{o}$ for rotations. This enables us to evaluate suitably defined functions f_{ij} and g_i considerably easier by use of an efficient data structure, called *no-fit polygons*, as will be discussed in Section 3. Given a solution $(\boldsymbol{x}, \boldsymbol{o})$, we fix the orientations $\boldsymbol{o}$, and introduce the following problem of reducing the total overlap translating polygons, which is called *polygon separation problem*:

$$\begin{aligned} &\text{minimize} \quad F(\boldsymbol{x}) = \sum_{1 \le i < j \le n} f_{ij}(\boldsymbol{x}) + \sum_{1 \le i \le n} g_i(\boldsymbol{x}) \\ &\text{subject to} \quad \boldsymbol{x} \in \mathbb{R}^{2n}, \end{aligned} \tag{4}$$

where we omit the indication of $\boldsymbol{o}$ for simplicity, We design a *separation algorithm* to the unconstrained nonlinear programming problem (4) by applying the *limited memory BFGS (L-BFGS) method* [13]. The algorithm translates all polygons simultaneously to construct a locally optimal solution.

Since the separation algorithm only translates polygons, we need a procedure for changing the orientations of polygons to handle (3). For this, we design a swapping procedure that changes the positions and orientations of two specified polygons to find their best positions and orientations under the condition that the positions and orientations of the other polygons are fixed.

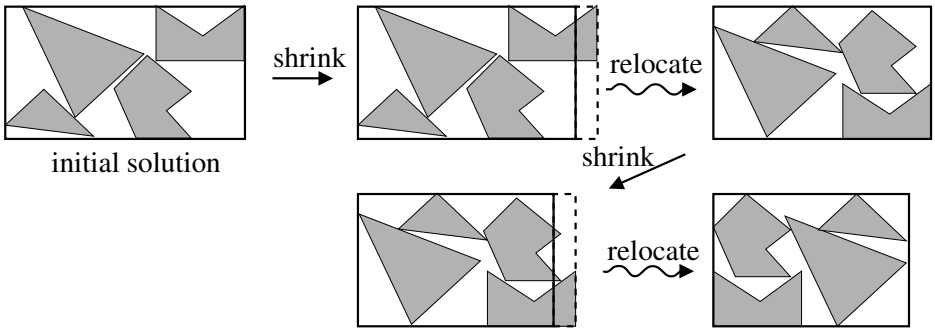

Fig. 2. Two layers of algorithm ILSQN

By combining the separation algorithm and the swapping procedure, we construct an iterated local search algorithm, called MINIMIZEOVERLAP, to find a solution to the overlap minimization problem (3). Given a layout $(\boldsymbol{x}, \boldsymbol{o})$, MINIMIZEOVERLAP$(\mathcal{P}, \mathcal{O}, C(W, L), \boldsymbol{x}, \boldsymbol{o})$ outputs a new layout $(\boldsymbol{x}^*, \boldsymbol{o}^*)$, which is obtained by modifying $(\boldsymbol{x}, \boldsymbol{o})$, and is a locally optimal solution to the problem (3). The details of MINIMIZEOVERLAP will be given in Section 4.

2.2 Entire Algorithm for the Irregular Strip Packing Problem

In this subsection, we give an entire description of algorithm ILSQN for problem (1). ILSQN first generates an initial solution by a method which we will give in Section 6.4, and sets the length L of the container so that $C(W, L)$ contains all polygons and the both left and right sides of $C(W, L)$ touch some polygons. Then ILSQN repeats two following layers of computations until a time limit is reached. One of the two layers is an outer layer that searches the minimum feasible length L^* by shrinking or extending the left and/or right sides of the container. For the current layout $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$, the outer layer chooses a length L of the container, where $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$ may be infeasible to the tentatively fixed length L. An inner layer, the other layer, then improves the current solution $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$ into a locally optimal solution for the new length L. To find such a solution, the inner layer invokes MINIMIZEOVERLAP. Figure 2 illustrates the behavior of the algorithm, where "shrink" corresponds to the outer layer and "relocate" corresponds to the inner layer.

We now explain how to execute the outer layer. The execution of the outer layer is described by parameters $r_{\text{dec}} \in (0, 1)$, $r_{\text{inc}} \in (0, 1)$ and $\pi_{\text{side}} \in \{\texttt{left}, \texttt{right}, \texttt{both}\}$. We shrink and extend the length L of the container by factors r_{dec} and r_{inc}, respectively. Parameter π_{side} determines which side of the container we shrink or extend. To be more precise, when ILSQN changes L to $L - l$, it translates the container to the right by l, 0, and $l/2$ if $\pi_{\text{side}} = \texttt{left}, \texttt{right}$, and $\texttt{both}$, respectively, as shown in Figure 3.

After computing an initial solution $(\boldsymbol{x}, \boldsymbol{o})$, we shorten L by $L := (1 - r_{\text{dec}})\mu(\boldsymbol{x}, \boldsymbol{o})$, and execute the inner layer. If the inner layer obtains a feasible layout successfully,

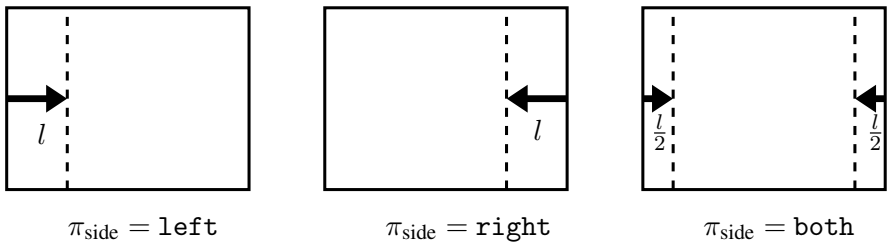

Fig. 3. Three ways of shrinking the container

then the outer layer shortens the length L of the container by $L := (1 - r_{\text{dec}})L$; otherwise, it extends the length by $L := (1 + r_{\text{inc}})L$. Algorithm ILSQN is formally described in Algorithm 1, in which we omit indication of π_{side} for simplicity.

Algorithm 1 : ILSQN($\mathcal{P}, \mathcal{O}, W$)

Generate an initial solution $(\boldsymbol{x}, \boldsymbol{o})$; {See Section 6.4}
$L_{\text{best}} := \mu(\boldsymbol{x}, \boldsymbol{o})$; $(\boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}}) := (\boldsymbol{x}, \boldsymbol{o})$;
$L := (1 - r_{\text{dec}})L_{\text{best}}$; $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}}) := (\boldsymbol{x}, \boldsymbol{o})$;
while within a time limit **do**
  $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}}) :=$ MINIMIZEOVERLAP$(\mathcal{P}, \mathcal{O}, C(W, L), \boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$; {See Algorithm 2}
  if $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$ is feasible **then**
    $L_{\text{best}} := L$; $(\boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}}) := (\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}})$;
    $L := (1 - r_{\text{dec}})L_{\text{best}}$
  else
    $L := (1 + r_{\text{inc}})L$;
    if $L \geq L_{\text{best}}$ **then**
      $L := (1 - r_{\text{dec}})L_{\text{best}}$; $(\boldsymbol{x}_{\text{cur}}, \boldsymbol{o}_{\text{cur}}) := (\boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}})$
    end if
  end if
end while;
Return $(L_{\text{best}}, \boldsymbol{x}_{\text{best}}, \boldsymbol{o}_{\text{best}})$

3 Computation of Overlap

This section defines suitable functions f_{ij} and g_i in the overlap minimization problem (3). Although there are several ways of defining these functions, we use the *penetration depth* (Section 3.2) to define them. The gradients of f_{ij} and g_i are important for our separation algorithm in Section 5 under the condition that all polygons' orientations are fixed. To compute the gradients of functions f_{ij} and g_i, we use the *no-fit polygons* (Section 3.1). We abbreviate $P_i(o_i)$ as P_i in Section 3.1, Section 3.2 and Section 3.3 for simplicity.

3.1 No-Fit Polygon

The *no-fit polygon* (NFP) is a data structure that is often used in algorithms for the irregular strip packing problem [2–5,7,8]. It is also used for other problems such as robotics, in which the no-fit polygon is called configuration-space obstacle.

The no-fit polygon $\mathrm{NFP}(P_i, P_j)$ for an ordered pair of two polygons P_i and P_j is defined by

$$\mathrm{NFP}(P_i, P_j) = \mathrm{int}(P_i) \oplus (-\mathrm{int}(P_j)) = \{\boldsymbol{v} - \boldsymbol{w} \mid \boldsymbol{v} \in \mathrm{int}(P_i),\ \boldsymbol{w} \in \mathrm{int}(P_j)\}.$$

The no-fit polygon has the following important properties:

- $P_i \oplus \boldsymbol{x}_i$ and $P_j \oplus \boldsymbol{x}_j$ overlap if and only if $\boldsymbol{x}_j - \boldsymbol{x}_i \in \mathrm{NFP}(P_i, P_j)$.
- $P_i \oplus \boldsymbol{x}_i$ touches $P_j \oplus \boldsymbol{x}_j$ if and only if $\boldsymbol{x}_j - \boldsymbol{x}_i \in \partial\,\mathrm{NFP}(P_i, P_j)$.
- $P_i \oplus \boldsymbol{x}_i$ and $P_j \oplus \boldsymbol{x}_j$ are separated if and only if $\boldsymbol{x}_j - \boldsymbol{x}_i \notin \mathrm{cl}(\mathrm{NFP}(P_i, P_j))$.

Hence the problem of checking whether two polygons overlap or not becomes an easier problem of checking whether a point is in a polygon or not. When P_i and P_j are both convex, $\partial\,\mathrm{NFP}(P_i, P_j)$ can be computed by the following simple procedure. We first place the reference point of P_i at the origin, and slide P_j around P_i, i.e., translate P_j having it keep touching with P_i. Then the trace of the reference point of P_j is $\partial\,\mathrm{NFP}(P_i, P_j)$. Practical algorithms to calculate an NFP of two non-convex polygons have been proposed, e.g., by Bennell et al. [14] and Ramkumar [15]. Figure 4 shows an example of $\mathrm{NFP}(P_i, P_j)$ of two polygons P_i and P_j where P_i is non-convex and P_j is convex.

We can also check whether a polygon P_i protrudes from the container C or not by using

$$\mathrm{NFP}(\overline{C}, P_i) = \mathrm{int}(\overline{C}) \oplus (-\mathrm{int}(P_i)) = \{\boldsymbol{v} - \boldsymbol{w} \mid \boldsymbol{v} \in \mathbb{R}^2 \setminus C,\ \boldsymbol{w} \in \mathrm{int}(P_i)\},$$

which is the complement of a rectangle whose boundary is the trajectory of the reference point of P_i when we slide P_i inside the container C. See an example in Figure 5. The following properties are derived from those of the no-fit polygon:

- $P_i \oplus \boldsymbol{x}_i$ protrudes from C if and only if $\boldsymbol{x}_i \in \mathrm{NFP}(\overline{C}, P_i)$.
- $P_i \oplus \boldsymbol{x}_i$ is contained in C and touches ∂C if and only if $\boldsymbol{x}_i \in \partial\,\mathrm{NFP}(\overline{C}, P_i)$.
- $P_i \oplus \boldsymbol{x}_i$ is contained in C and does not touch ∂C if and only if $\boldsymbol{x}_i \notin \mathrm{cl}(\mathrm{NFP}(\overline{C}, P_i))$.

To check if a polygon $P_i \oplus \boldsymbol{x}_i$ protrudes from C, Gomes and Oliveira [4,8] introduced the *inner-fit rectangle*, which is equivalent to $\overline{\mathrm{NFP}(\overline{C}, P_i)}$.

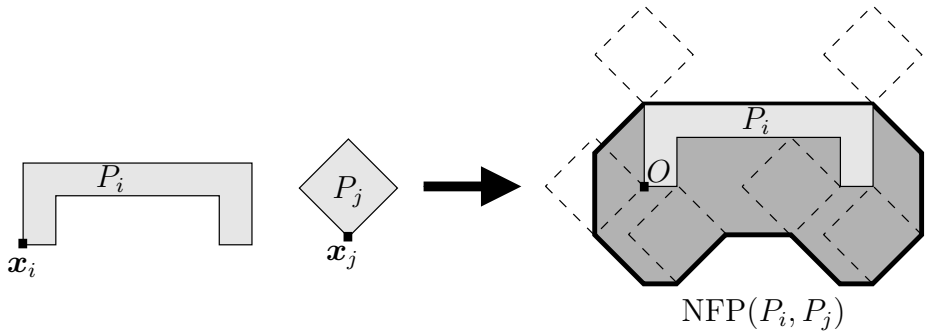

Fig. 4. Illustration of $\mathrm{NFP}(P_i, P_j)$ (O is the origin)

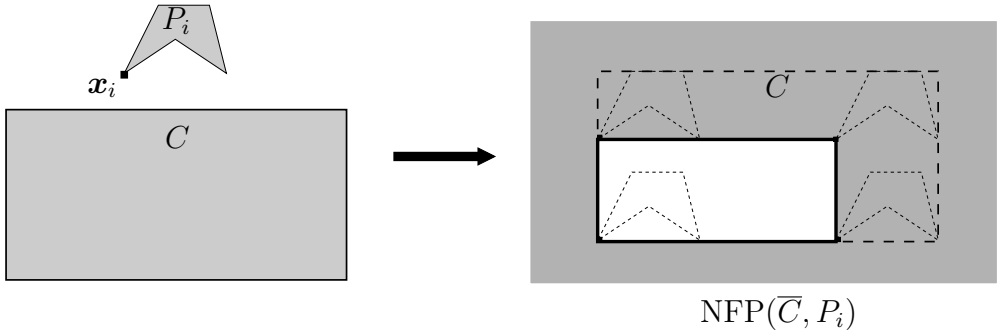

Fig. 5. Illustration of $\mathrm{NFP}(\overline{C}, P_i)$

3.2 Penetration Depth

The *penetration depth* (also known as the intersection depth) is an important notion used for robotics, computer vision and so on [16–18]. The penetration depth $\delta(P_i, P_j)$ of two overlapping polygons P_i and P_j is defined to be the minimum translational distance to separate them. If two polygons do not overlap, their penetration depth is defined to be zero. Formally, the penetration depth $\delta(P_i, P_j)$ of two polygons P_i and P_j is defined by

$$\delta(P_i, P_j) = \min\{\|\boldsymbol{z}\| \mid \mathrm{int}(P_i) \cap (P_j \oplus \boldsymbol{z}) = \emptyset,\ \boldsymbol{z} \in \mathbb{R}^2\},$$

where $\|\cdot\|$ denotes the Euclidean norm.

Figure 6 shows the relationship between the penetration depth and the NFP. We can separate two polygons P_i and P_j by translating the reference point of P_j to a point $\boldsymbol{x}'$ on $\partial\,\mathrm{NFP}(P_i, P_j)$. Hence $\delta(P_i \oplus \boldsymbol{x}_i, P_j \oplus \boldsymbol{x}_j)$ is the minimum distance from $\boldsymbol{x}_j - \boldsymbol{x}_i$ to $\partial\,\mathrm{NFP}(P_i, P_j)$. The solid and dashed arrows are examples of translations of P_j to the boundary of the NFP and the dashed polygons are the polygons of P_j translated by the vectors represented by these arrows. The solid arrow $\boldsymbol{x}'' - (\boldsymbol{x}_j - \boldsymbol{x}_i)$ has the minimum distance among all translations, giving the penetration depth $\delta(P_i \oplus \boldsymbol{x}_i, P_j \oplus \boldsymbol{x}_j)$.

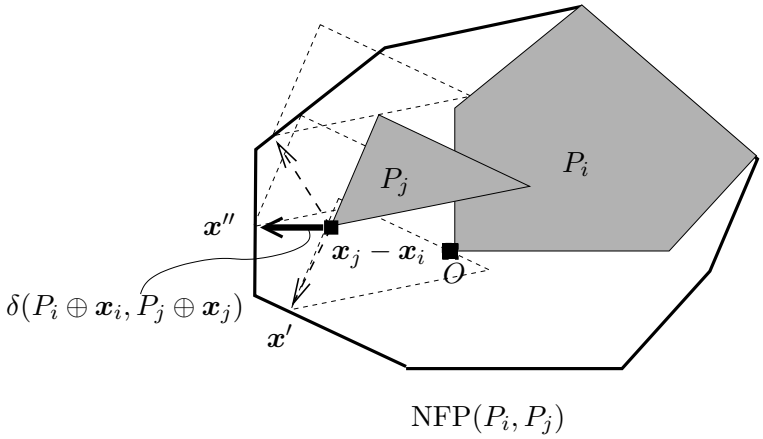

Fig. 6. The no-fit polygon $\mathrm{NFP}(P_i, P_j)$ and the penetration depth $\delta(P_i \oplus \boldsymbol{x}_i, P_j \oplus \boldsymbol{x}_j)$

3.3 Amount of Overlap

We define functions f_{ij} and g_i in problem (3) using the penetration depth. To represent the amount of overlap between P_i and P_j, we define f_{ij} by

$$f_{ij}(\boldsymbol{x}) = \delta(P_i \oplus \boldsymbol{x}_i, P_j \oplus \boldsymbol{x}_j)^m, \quad 1 \leq i < j \leq n,$$

where $\boldsymbol{x} = (\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n)$ and m is a positive parameter. Similarly we define $g_i(\boldsymbol{x})$ by

$$g_i(\boldsymbol{x}) = \delta(\mathrm{cl}(\overline{C}), P_i \oplus \boldsymbol{x}_i)^m, \quad 1 \leq i \leq n.$$

In order to apply efficient algorithms for solving the nonlinear program to the polygon separation problem (4), we need to compute the values of $f_{ij}(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$ and their gradients for a given solution $(\boldsymbol{x}, \boldsymbol{o})$, where all polygons' orientations $\boldsymbol{o}$ are fixed. We describe below how we realize such computation. Let $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ be the translation vectors of P_i and P_j, respectively, and denote $\boldsymbol{v} = \boldsymbol{x}_j - \boldsymbol{x}_i$ for convenience. We first consider how to compute $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$, and later explain the case of $g_i(\boldsymbol{x})$ and $\nabla g_i(\boldsymbol{x})$. There are three cases for the computation of $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$.

Case 1: two polygons P_i and P_j do not overlap. In this case, we see that $f_{ij}(\boldsymbol{x}) = 0$ and $\nabla f_{ij}(\boldsymbol{x}) = \boldsymbol{0}$.

Case 2: two polygons P_i and P_j overlap (i.e., $f_{ij}(\boldsymbol{x}) > 0$) and the nearest point on $\partial\,\mathrm{NFP}(P_i, P_j)$ from $\boldsymbol{v}$ is unique. See an example in Figure 7. Let $\boldsymbol{w}$ be the nearest point and let $\boldsymbol{z} = \boldsymbol{w} - \boldsymbol{v}$. Because the variable $\boldsymbol{x}$ is a list of n 2-dimensional vectors, $\nabla f_{ij}(\boldsymbol{x})$ is a list of the same size; hence we denote $\nabla f_{ij}(\boldsymbol{x}) = (\nabla_1 f_{ij}(\boldsymbol{x}), \ldots, \nabla_n f_{ij}(\boldsymbol{x}))$, where $\nabla_k = (\partial/\partial x_{k1}, \partial/\partial x_{k2})$, $1 \leq k \leq n$. Then, $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$ for $1 \leq i <$

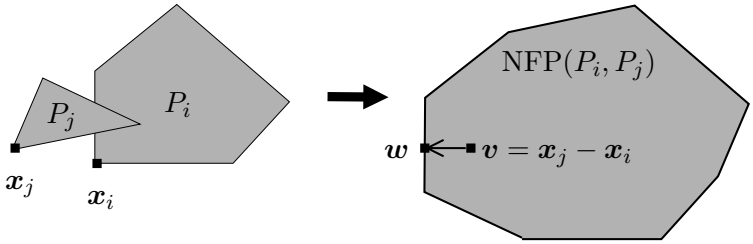

Fig. 7. The computation of $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$

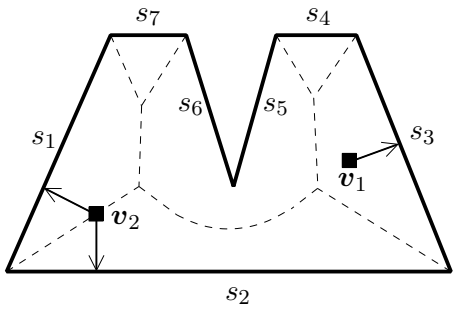

Fig. 8. The medial axis of an NFP and the nearest points on the boundary from inner points $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$

$j \leq n$ are given by

$$\begin{aligned} f_{ij}(\boldsymbol{x}) &= \|\boldsymbol{z}\|^m, \\ \nabla_i f_{ij}(\boldsymbol{x}) &= -\nabla_j f_{ij}(\boldsymbol{x}) = m\|\boldsymbol{z}\|^{m-2}\boldsymbol{z}, \\ \nabla_k f_{ij}(\boldsymbol{x}) &= \boldsymbol{0}, \quad k \in \{1, \ldots, n\} \setminus \{i, j\}. \end{aligned} \tag{5}$$

Every $\nabla_k f_{ij}(\boldsymbol{x})$ except $\nabla_i f_{ij}(\boldsymbol{x})$ and $\nabla_j f_{ij}(\boldsymbol{x})$ is zero because only P_i and P_j can contribute to the overlap $f_{ij}(\boldsymbol{x})$.

Case 3: $f_{ij}(\boldsymbol{x}) > 0$ and the nearest point from $\boldsymbol{v}$ to $\partial\,\mathrm{NFP}(P_i, P_j)$ is not unique. In this case, ∇f_{ij} is not differentiable at $\boldsymbol{x}$; however, we choose one of the nearest points arbitrarily as $\boldsymbol{w}$ and calculate $\nabla f_{ij}(\boldsymbol{x})$ with (5) as in Case 2.

Case 2 and Case 3 are distinguished in reference to the *medial axis* [19,20] of $\mathrm{NFP}(P_i, P_j)$. The medial axis of a polygon P is defined by the trace of the centers of all circles contained in P that touch at least two sides of ∂P. Figure 8 shows an example of an NFP and its medial axis. The thick solid polygon is an NFP, $s_1, \ldots, s_7$ are the edges of the NFP, and the dashed lines are the medial axis of the NFP. For the two points $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, the arrows indicate the nearest point on the NFP from $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$, respectively. The nearest point on the NFP's boundary from a point $\boldsymbol{v}$ in the NFP is unique if and only if $\boldsymbol{v}$ is not on the medial axis of the NFP. In Figure 8, $\boldsymbol{v}_1$ is not on the medial axis and it has the unique nearest point on s_3. On the other hand $\boldsymbol{v}_2$ is on the medial axis and it has two nearest points on s_1 and s_2. Note that $\boldsymbol{v}$ can have more than one nearest point only when $\boldsymbol{v}$ is on the medial axis of the NFP. Such a case is rare because the search basically tries to change $\boldsymbol{v}$ so that it moves away from the medial axis in order to minimize the sum of $f_{ij}(\boldsymbol{x})$.

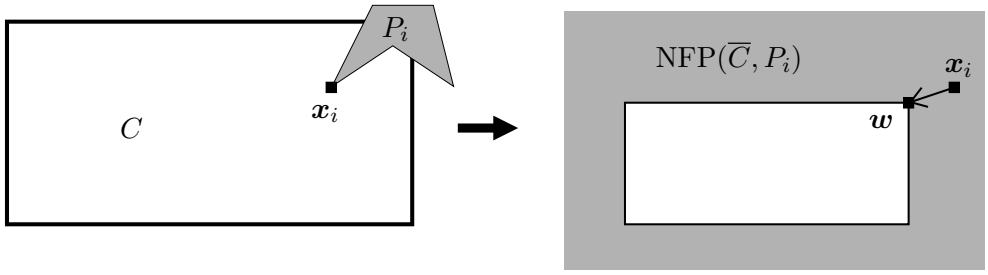

Fig. 9. The computation of $g_i(\boldsymbol{x})$ and $\nabla g_i(\boldsymbol{x})$

We compute $g_i(\boldsymbol{x})$ and $\nabla g_i(\boldsymbol{x})$ similarly as in the case of $f_{ij}(\boldsymbol{x})$ and $\nabla f_{ij}(\boldsymbol{x})$. If P_i is contained in C, we simply return $g_i(\boldsymbol{x}) = 0$ and $\nabla g_i(\boldsymbol{x}) = \boldsymbol{0}$. We consider a polygon P_i that protrudes from the container C (i.e., $g_i(\boldsymbol{x}) > 0$). See an example in Figure 9. Let $\boldsymbol{w}$ be the nearest point on $\partial\,\mathrm{NFP}(\overline{C}, P_i)$ from $\boldsymbol{x}_i$ and $\boldsymbol{z} = \boldsymbol{w} - \boldsymbol{x}_i$; the nearest point is always unique in this case. We denote $\nabla g_i(\boldsymbol{x}) = (\nabla_1 g_i(\boldsymbol{x}), \ldots, \nabla_n g_i(\boldsymbol{x}))$. Then, $g_i(\boldsymbol{x})$ and its gradient for $1 \leq i < j \leq n$ are given by

$$\begin{aligned} g_i(\boldsymbol{x}) &= \|\boldsymbol{z}\|^m, \\ \nabla_i g_i(\boldsymbol{x}) &= -m\|\boldsymbol{z}\|^{m-2}\boldsymbol{z}, \\ \nabla_k g_i(\boldsymbol{x}) &= \boldsymbol{0}, \quad k \in \{1, \ldots, n\} \setminus \{i\}. \end{aligned} \tag{6}$$

Every $\nabla_k g_i(\boldsymbol{x})$ except $\nabla_i g_i(\boldsymbol{x})$ is zero because only P_i has influence on its protrusion from the container.

In Case 2, $f_{ij}(\boldsymbol{x})$ is not differentiable at $\boldsymbol{x}$ if and only if P_i and P_j touch each other and $0 < m \leq 1$. Similarly $g_i(\boldsymbol{x})$ is not differentiable at $\boldsymbol{x}$ if and only if P_i is contained in C, touches C and $0 < m \leq 1$. We avoid choosing such m as will be discussed below. We can thus calculate the gradient of the objective function of (3).

The positive parameter m determines the differentiability of $f_{ij}(\boldsymbol{x})$ and $g_i(\boldsymbol{x})$. Figure 10 shows the change of $f_{ij}(\boldsymbol{x})$ along the arrow from the outside to the inside of an NFP. At the boundary of the NFP, $f_{ij}(\boldsymbol{x})$ is not differentiable for $0 < m \leq 1$, while it is differentiable for $m > 1$. Moreover, $\nabla f_{ij}(\boldsymbol{x})$ in (5) becomes simpler for $m = 2$ because term $\|z\|^{m-2}$ disappears. The situation is the same for $g_i(\boldsymbol{x})$, and hence we let $m = 2$ in our experiments.

3.4 Computing NFPs for ILSQN

In the previous subsections, we show how to use no-fit polygons to evaluate functions f_{ij} and g_i and their gradients, where the orientations of polygons are fixed for simplicity. However, in our algorithm ILSQN, we need to use no-fit polygons $\mathrm{NFP}(P_i(o_i), P_j(o_j))$ and $\mathrm{NFP}(\overline{C}, P_i(o_i))$ for all possible orientations o_i and o_j. We thus compute all $\mathrm{NFP}(P_i(o_i), P_j(o_j))$, $o_i \in O_i$, $o_j \in O_j$, $1 \leq i < j \leq n$ beforehand and utilize them in ILSQN.

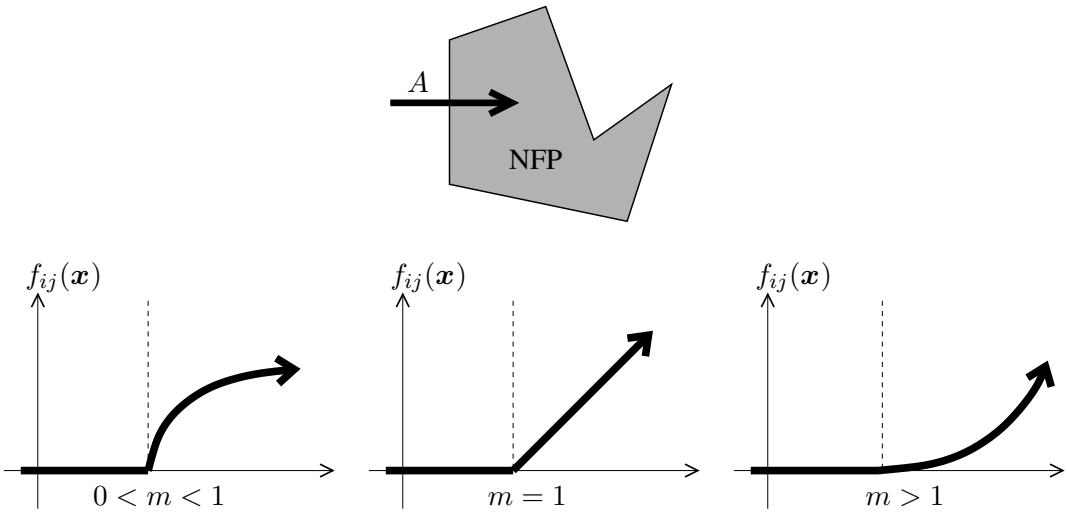

Fig. 10. The value of $f_{ij}(\boldsymbol{x})$ on arrow A

4 Iterated Local Search for the Overlap Minimization Problem

In this section, we formally describe MINIMIZEOVERLAP, our iterated local search algorithm for the overlap minimization problem (3) introduced in Section 2.1. MINIMIZEOVERLAP invokes the following algorithms.

- SEPARATE: an algorithm that translates all polygons in a given layout simultaneously to reduce the total amount of overlap and protrusion, where the length L of the container is fixed. It does not consider rotation. The detail is given in Section 5.
- SWAPTWOPOLYGONS: an operation of swapping two specified polygons in a given layout considering rotations, where the other polygons and the length L of the container are fixed. The detail is given in Section 6.

MINIMIZEOVERLAP maintains the earliest solution that minimizes the objective function F of (3) among those searched by then as the incumbent solution $(\boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}})$, which will be used for generating the next initial solution. MINIMIZEOVERLAP first perturbs the incumbent solution by swapping two randomly chosen polygons P_i and P_j calling SWAPTWOPOLYGONS$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}}, P_i, P_j)$, and then translates all polygons simultaneously calling SEPARATE$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}_{\text{init}}, \boldsymbol{o}_{\text{init}})$ to obtain a locally optimal solution $(\boldsymbol{x}_{\text{lopt}}, \boldsymbol{o}_{\text{lopt}})$ of (3). Since SEPARATE does not rotate polygons, $\boldsymbol{o}_{\text{lopt}} = \boldsymbol{o}_{\text{init}}$ holds. If the locally optimal solution has less overlap than the incumbent solution does, we update the incumbent solution with the locally optimal solution. MINIMIZEOVERLAP stops these operations if it fails to update the incumbent solution after a prescribed number N_{mo} of consecutive calls to local search. The algorithm is formally described in Algorithm 2, where $(\boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}})$ is an initial layout given to the algorithm.

Algorithm 2 : MINIMIZEOVERLAP$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}})$

$k := 0$;
while $k < N_{\text{mo}}$ **do**
  Randomly choose P_i and P_j from $\mathcal{P}$;
  $(\boldsymbol{x}_{\text{init}}, \boldsymbol{o}_{\text{init}}) :=$ SWAPTWOPOLYGONS$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}}, P_i, P_j)$;
  $(\boldsymbol{x}_{\text{lopt}}, \boldsymbol{o}_{\text{lopt}}) :=$ SEPARATE$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}_{\text{init}}, \boldsymbol{o}_{\text{init}})$;
  $k := k + 1$;
  if $F(\boldsymbol{x}_{\text{lopt}}, \boldsymbol{o}_{\text{lopt}}) < F(\boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}})$ **then**
    $(\boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}}) := (\boldsymbol{x}_{\text{lopt}}, \boldsymbol{o}_{\text{lopt}})$;
    $k := 0$
  end if
end while;
Return $(\boldsymbol{x}_{\text{inc}}, \boldsymbol{o}_{\text{inc}})$

5 Separation Algorithm

This section describes our separation algorithm SEPARATE, which is used as local search in MINIMIZEOVERLAP. The polygon separation problem (4) introduced in Section 2.1 is an unconstrained nonlinear programming problem, where all polygons' orientations $\boldsymbol{o}$ and the length L of the container are fixed. The objective function F of (4) and its gradient ∇F are efficiently computable because ∇f_{ij} and ∇g_i are computable with the no-fit polygons as described in Section 3.3. SEPARATE$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o})$ is thus realized as follows: SEPARATE first applies the L-BFGS method to the polygon separation problem (4) by using the current layout $(\boldsymbol{x}, \boldsymbol{o})$ as an initial solution and SEPARATE returns a locally optimal solution delivered by the L-BFGS method. SEPARATE translates all polygons simultaneously to reduce the total amount of overlap and protrusion, where no rotation is considered. The idea is natural; however, to the best of our knowledge, it seems that nonlinear programming approaches of this type have not been successfully applied.

6 Swapping Two Polygons

6.1 Moving a Polygon

ILSQN perturbs a locally optimal solution by swapping two polygons in the solution. Instead of just exchanging two polygons P_i and P_j in their reference points, we attempt to find their positions with the least overlap. FINDBESTPOSITION$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_i)$ is a heuristic algorithm to find a minimum overlap position of a specified polygon P_i without changing the positions of the other polygons, while considering all possible orientations $o \in O_i$ of P_i. Let $\mathcal{N}(o)$ be the set of polygon boundaries $\partial\,\mathrm{NFP}(P_k(o_k) \oplus \boldsymbol{x}_k, P_i(o))$, $P_k \in \mathcal{P} \setminus \{P_i\}$ and $\partial\,\mathrm{NFP}(\overline{C}, P_i(o))$, $\mathcal{V}(o)$

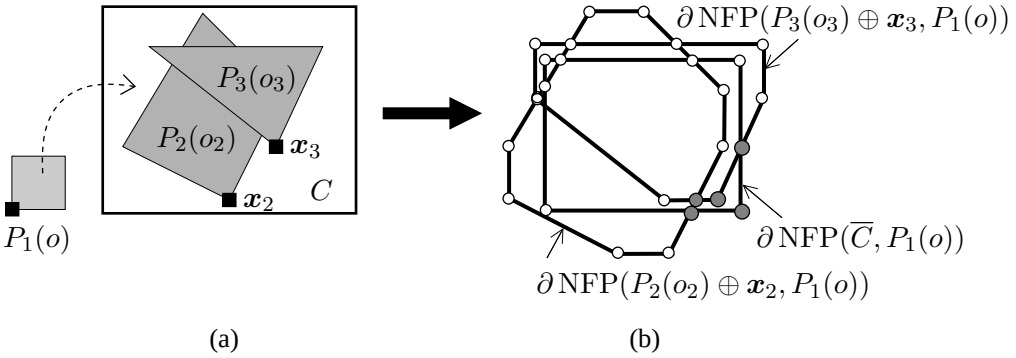

Fig. 11. (a) A layout of P_2 and P_3 in C; (b) $\partial\,\mathrm{NFP}(P_i(o_i) \oplus \boldsymbol{x}_i, P_1(o))$ $(i = 2, 3)$ and $\partial\,\mathrm{NFP}(\overline{C}, P_1(o))$

be the set of vertices of $\mathcal{N}(o)$, and $\mathcal{I}(o)$ be the set of edge intersections of $\mathcal{N}(o)$. For each point $\boldsymbol{v} \in \mathcal{V}(o) \cup \mathcal{I}(o)$, the heuristics computes the overlap of $P_i(o) \oplus \boldsymbol{v}$ with the other polygons, and finds the position with the least overlap, where $\boldsymbol{x}$ is a list of the translation vectors of polygons, $\boldsymbol{o}$ is a list of the orientations of polygons, and the amount of overlap is computed by the objective function $F(\boldsymbol{x}, \boldsymbol{o})$ of the overlap minimization problem (3). By repeating these operations for all orientations $o \in O_i$ of P_i, FINDBESTPOSITION seeks the best position and orientation. The algorithm is formally described in Algorithm 3, where $(\boldsymbol{x})_i$ denotes the ith element of $\boldsymbol{x}$ and $(\boldsymbol{o})_i$ denotes the ith element of $\boldsymbol{o}$.

Algorithm 3 : FINDBESTPOSITION$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_i)$

```
F* := +∞;
for each o ∈ O_i do
  Compute NFP(P_k(o_k) ⊕ x_k, P_i(o)) (P_k ∈ P \ {P_i}) and NFP(C̄, P_i(o));
  for each v ∈ V(o) ∪ I(o) do
    x' := x; (x')_i := v;
    o' := o; (o')_i := o;
    if F(x', o') < F* then
      F* := F(x', o'); v* := v; o* := o
    end if
  end for
end for;
Return (v*, o*)
```

Figure 11 shows an example in which FINDBESTPOSITION is searching for the best position for a square $P_1(o)$ with a fixed orientation $o \in O_1$ in the left layout. Figure 11(b) shows $\mathrm{NFP}(P_2(o_2) \oplus \boldsymbol{x}_2, P_1(o))$, $\mathrm{NFP}(P_3(o_3) \oplus \boldsymbol{x}_3, P_1(o))$, and $\mathrm{NFP}(\overline{C}, P_1(o))$. The circles in Figure 11(b) represent $\mathcal{V}(o)$ and $\mathcal{I}(o)$. FINDBESTPOSITION finds a point that corresponds to a layout with the least overlap. Each grey circle in Figure 11(b) corresponds to a position that has no overlap in Figure 11(a).

FINDBESTPOSITION has an important property described in Theorem 1.

Theorem 1 FINDBESTPOSITION$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_i)$ *always finds a point* $\hat{\boldsymbol{v}} \in \mathbb{R}^2$ *and an orientation* $\hat{o} \in O_i$ *of polygon* P_i *such that* $P_i(\hat{o}) \oplus \hat{\boldsymbol{v}}$ *neither overlaps with the other polygons nor protrudes from the container* C *if there exists such a pair of a point and an orientation.*

Proof Assume that there exists a point $\hat{\boldsymbol{v}} \in \mathbb{R}^2$ and an orientation $\hat{o} \in O_i$ of polygon P_i such that $P_i(\hat{o}) \oplus \hat{\boldsymbol{v}}$ neither overlaps with the other polygons nor protrudes from the container C. Since FINDBESTPOSITION tries all orientations in O_i, it chooses $o = \hat{o}$ in the outer for-loop.

Note that $P_i(o) \oplus \boldsymbol{v}$ does not overlap with $P_k(o_k) \oplus \boldsymbol{x}_k$ if and only if $\boldsymbol{v} \in \overline{\mathrm{NFP}(P_k(o_k) \oplus \boldsymbol{x}_k, P_i(o))}$, and $P_i(o) \oplus \boldsymbol{v}$ does not protrude from the container C if and only if $\boldsymbol{v} \in \overline{\mathrm{NFP}(\overline{C}, P_i(o))}$. Hence, for $\hat{o}$, the set Z of all points $\boldsymbol{v}$ such that $P_i(\hat{o}) \oplus \boldsymbol{v}$ neither overlaps with the other polygon nor protrudes from the container C is given by

$$Z = \bigcap_{P_k \in \mathcal{P} \setminus \{P_i\}} \overline{\mathrm{NFP}(P_k(o_k) \oplus \boldsymbol{x}_k, P_i(\hat{o}))} \cap \overline{\mathrm{NFP}(\overline{C}, P_i(\hat{o}))}.$$

We see that Z is closed and bounded because $\overline{\mathrm{NFP}(P_k(o_k) \oplus \boldsymbol{x}_k, P_i(\hat{o}))}$ $(P_k \in \mathcal{P} \setminus \{P_i\})$ and $\overline{\mathrm{NFP}(\overline{C}, P_i(\hat{o}))}$ are all closed, and $\overline{\mathrm{NFP}(\overline{C}, P_i(\hat{o}))}$ is bounded. Since Z is not empty by the assumption, $\partial Z \subseteq Z$ is not empty either. Z is surrounded by some edges of $\partial\,\mathrm{NFP}(P_k(o_k) \oplus \boldsymbol{x}_k, P_i(\hat{o}))$ and $\partial\,\mathrm{NFP}(\overline{C}, P_i(\hat{o}))$. Therefore, ∂Z includes a point in $\mathcal{V}(o) \cup \mathcal{I}(o)$ and hence FINDBESTPOSITION will choose such a point.

For any $(\hat{\boldsymbol{v}}, \hat{o})$ such that $P_i(\hat{o}) \oplus \hat{\boldsymbol{v}}$ neither overlaps with the other polygons nor protrudes from the container, $F(\boldsymbol{x}, \boldsymbol{o})$ is strictly smaller than any other $(\boldsymbol{v}, o)$ such that $P_i(o) \oplus \boldsymbol{v}$ overlaps with another polygon or protrudes from the container, since $F(\boldsymbol{x}, \boldsymbol{o})$ at $(\hat{\boldsymbol{v}}, \hat{o})$ is the sum of the amount of overlap and protrusion of all polygons except P_i. Therefore FINDBESTPOSITION will output such a $(\hat{\boldsymbol{v}}, \hat{o})$ if any. □

However, FINDBESTPOSITION may not find the globally optimal position if there is no position whose overlap is zero.

6.2 Simplifying FINDBESTPOSITION

We observed through preliminary experiments that it is time consuming to compute the objective function $F(\boldsymbol{x}, \boldsymbol{o})$ of (3) for all points in $\mathcal{V}(o)$ and $\mathcal{I}(o)$ in FINDBESTPOSITION. We therefore simplify FINDBESTPOSITION by reducing the candidates of $\boldsymbol{v}$ for which we compute $F(\boldsymbol{x}, \boldsymbol{o})$. FINDBESTPOSITION2 is the simplified version of FINDBESTPOSITION. It checks all possible orientations $o \in O_i$, but it

checks them for a prescribed number K of points randomly chosen from $\mathcal{V}(o)$ and for no point in $\mathcal{I}(o)$. The algorithm FINDBESTPOSITION2 is formally described in Algorithm 4. FINDBESTPOSITION2 does not satisfy Theorem 1.

Algorithm 4 : FINDBESTPOSITION2$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_i)$

$F^* := +\infty$;
for each $o \in O_i$ **do**
  Compute $\mathrm{NFP}(P_k \oplus \boldsymbol{x}_k, P_i(o))$ $(P_k \in \mathcal{P} \setminus \{P_i\})$ and $\mathrm{NFP}(\overline{C}, P_i(o))$;
  Let V be a set of K vertices randomly chosen from $\mathcal{V}(o)$;
  for each $\boldsymbol{v} \in V$ **do**
    $\boldsymbol{x}' := \boldsymbol{x}$; $(\boldsymbol{x}')_i := \boldsymbol{v}$;
    $\boldsymbol{o}' := \boldsymbol{o}$; $(\boldsymbol{o}')_i := o$;
    if $F(\boldsymbol{x}', \boldsymbol{o}') < F^*$ **then**
      $F^* := F(\boldsymbol{x}', \boldsymbol{o}')$; $\boldsymbol{v}^* := \boldsymbol{v}$; $o^* := o$
    end if
  end for
end for;
Return $(\boldsymbol{v}^*, o^*)$

6.3 *Swapping Two Polygons*

SWAPTWOPOLYGONS$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_i, P_j)$ is an algorithm to swap two polygons P_i and P_j. We designed three ways of swapping, which are described by a parameter $\sigma \in \{\mathtt{C}, \mathtt{I}\} \cup \mathbb{N}$, where $\mathbb{N}$ denotes the set of natural numbers. When $\sigma = \mathtt{C}$, we swap the centroids of P_i and P_j (it is only for comparisons in Section 7.2). When $\sigma = \mathtt{I}$, we first remove the polygon P_i from the container C, which results in making a hole in the layout. We next place a new polygon $P'(o_j) \oplus \boldsymbol{x}_j$, where $P' = P_j$, to prevent P_j from staying at the same place. Then we move P_j to the position computed by FINDBESTPOSITION$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_j)$ and remove P', where we expect that P_j moves into the hole. Finally, we place the removed polygon P_i by FINDBESTPOSITION. When $\sigma = \mathbb{N}$, we move polygons as the case of $\sigma = \mathtt{I}$, but we compute the positions of P_i and P_j by FINDBESTPOSITION2$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_j)$, where we set $K = \sigma$. The algorithm SWAPTWOPOLYGONS is formally described in Algorithm 5, where $(\mathcal{P})_i$ denotes the ith element of $\mathcal{P}$ and $(\mathcal{O})_i$ denotes the ith element of $\mathcal{O}$. We compare the three ways of swapping by computational experiments in Section 7.2.

6.4 *Initial Solution of ILSQN*

We generate an initial feasible layout of ILSQN using FINDBESTPOSITION or FINDBESTPOSITION2. We assume that the initial length L_0 of the container $C(W, L)$

Algorithm 5 : SWAPTWOPOLYGONS($\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, P_i, P_j$)

{For $\sigma = \mathtt{C}$}
for each $(o'_i, o'_j) \in O_i \times O_j$ **do**
  Swap $P_i(o'_i)$ and $P_j(o'_j)$ at their centroids;
  Let $(\boldsymbol{x}', \boldsymbol{o}')$ be the resulting layout
end for;
Let $(\boldsymbol{x}, \boldsymbol{o})$ be the $(\boldsymbol{x}', \boldsymbol{o}')$ with the minimum $F(\boldsymbol{x}', \boldsymbol{o}')$ among those generated in the above loop;
Return $(\boldsymbol{x}, \boldsymbol{o})$

{For $\sigma = \mathtt{I}$}
$\mathcal{P}' := \mathcal{P}$; $\mathcal{O}' := \mathcal{O}$; $\boldsymbol{x}' := \boldsymbol{x}$; $\boldsymbol{o}' := \boldsymbol{o}$;
$(\mathcal{P}')_i := (\mathcal{P})_j$; $(\mathcal{O}')_i := (\boldsymbol{o})_j$; $(\boldsymbol{x}')_i := (\boldsymbol{x})_j$; $(\boldsymbol{o}')_i := (\boldsymbol{o})_j$;
$((\boldsymbol{x})_j, (\boldsymbol{o})_j) :=$ FINDBESTPOSITION$(\mathcal{P}', \mathcal{O}', C, \boldsymbol{x}', \boldsymbol{o}', (\mathcal{P}')_j)$;
$((\boldsymbol{x})_i, (\boldsymbol{o})_i) :=$ FINDBESTPOSITION$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, (\mathcal{P})_i)$;
Return $(\boldsymbol{x}, \boldsymbol{o})$

{For $\sigma \in \mathbb{N}$}
$\mathcal{P}' := \mathcal{P}$; $\mathcal{O}' := \mathcal{O}$; $\boldsymbol{x}' := \boldsymbol{x}$; $\boldsymbol{o}' := \boldsymbol{o}$;
$(\mathcal{P}')_i := (\mathcal{P})_j$; $(\mathcal{O}')_i := (\boldsymbol{o})_j$; $(\boldsymbol{x}')_i := (\boldsymbol{x})_j$; $(\boldsymbol{o}')_i := (\boldsymbol{o})_j$;
$((\boldsymbol{x})_j, (\boldsymbol{o})_j) :=$ FINDBESTPOSITION2$(\mathcal{P}', \mathcal{O}', C, \boldsymbol{x}', \boldsymbol{o}', (\mathcal{P}')_j)$;
$((\boldsymbol{x})_i, (\boldsymbol{o})_i) :=$ FINDBESTPOSITION2$(\mathcal{P}, \mathcal{O}, C, \boldsymbol{x}, \boldsymbol{o}, (\mathcal{P})_i)$;
Return $(\boldsymbol{x}, \boldsymbol{o})$

is large enough to place all polygons in $\mathcal{P}$ without overlap in C. We let $L_0 = 10^9$ in our experiments. We prepare a sequence of polygons in $\mathcal{P}$ and place polygons one by one in the order of the sequence, where the position of each polygon P_i is decided by FINDBESTPOSITION or FINDBESTPOSITION2. We control the sequence by using a parameter $S_{\text{init}} \in \{\mathtt{sort}, \mathtt{random}\}$: the sequence of the descending order of area if $S_{\text{init}} = \mathtt{sort}$ and a random sequence if $S_{\text{init}} = \mathtt{random}$. We use FINDBESTPOSITION if $\sigma \in \{\mathtt{C}, \mathtt{I}\}$ and FINDBESTPOSITION2 if $\sigma \in \mathbb{N}$, where we give σ to FINDBESTPOSITION2 as the parameter K. If there are more than one position with no overlap, we choose the bottom-left position (i.e., the position with the minimum x_{i1}, breaking ties with the minimum x_{i2}, where $\boldsymbol{x}_i = (x_{i1}, x_{i2})$ is the translation vector of polygon P_i). We compute the length L of the container by (2) after placing all polygons in $\mathcal{P}$.

There are variations in choosing a sequence of polygons such as the descending order of area and a random order. Gomes and Oliveira [8] also generated initial solutions in a similar way using a different sequence of polygons.

Table 1
Information of instances (cited from [8])

Instance	NDP	TNP	ANV	Orientations (°)	Width
ALBANO	8	24	7.25	0, 180	4900
DAGLI	10	30	6.30	0, 180	60
DIGHE1	16	16	3.87	0	100
DIGHE2	10	10	4.70	0	100
FU	12	12	3.58	0, 90, 180, 270	38
JAKOBS1	25	25	5.60	0, 90, 180, 270	40
JAKOBS2	25	25	5.36	0, 90, 180, 270	70
MAO	9	20	9.22	0, 90, 180, 270	2550
MARQUES	8	24	7.37	0, 90, 180, 270	104
SHAPES0	4	43	8.75	0	40
SHAPES1	4	43	8.75	0, 180	40
SHAPES2	7	28	6.29	0, 180	15
SHIRTS	8	99	6.63	0, 180	40
SWIM	10	48	21.90	0, 180	5752
TROUSERS	17	64	5.06	0, 180	79

NDP: The number of different polygons
TNP: The total number of polygons
ANV: The average number of vertices of different polygons

7 Computational Results

This section reports the results on computational experiments of our algorithm ILSQN and other algorithms.

7.1 Environment

Benchmark instances for the irregular strip packing problem are available online at EURO Special Interest Group on Cutting and Packing (ESICUP) website [1] . Table 1 shows the information of the instances. The column NDP shows the number of different polygons, the column TNP shows the total number of polygons, the column ANV shows the average number of the vertices of different polygons, the column Orientations shows the permitted orientations, and the column Width shows the width W of the container.

We implemented our algorithm ILSQN in C++, compiled it by GCC 4.0.2 and con-

[1] ESICUP: `http://paginas.fe.up.pt/~esicup/tiki-index.php`

ducted computational experiments on a PC with an Intel Xeon 2.8GHz processor (NetBurst microarchitecture) and 1GB memory. The L-BFGS method has a parameter m_{BFGS} that is the number of BFGS corrections. We set $m_{\mathrm{BFGS}} = 6$ because $3 \leq m_{\mathrm{BFGS}} \leq 7$ is recommended by Liu and Nocedal [13].

A layout is judged to be feasible when the objective function $F(\boldsymbol{x}, \boldsymbol{o})$ of (3) is less than $\epsilon = 10^{-10}W^2$ due to limited precision. Thus, our algorithm may generate layouts that have a slight overlap.

7.2 Parameters

ILSQN has the following parameters:

- $r_{\mathrm{dec}} \in (0, 1)$: the ratio by which ILSQN shortens the container.
- $r_{\mathrm{inc}} \in (0, 1)$: the ratio by which ILSQN extends the container.
- $\pi_{\mathrm{side}} \in \{\mathtt{left}, \mathtt{right}, \mathtt{both}\}$: the side of the container ILSQN shortens or extends.
- $N_{\mathrm{mo}} > 0$: the termination criterion of MINIMIZEOVERLAP.
- $S_{\mathrm{init}} \in \{\mathtt{sort}, \mathtt{random}\}$: the sequence from which ILSQN generates an initial solution. "$S_{\mathrm{init}} = \mathtt{sort}$" means the descending order of area and "$S_{\mathrm{init}} = \mathtt{random}$" means a random sequence.
- $\sigma \in \{\mathtt{C}, \mathtt{I}\} \cup \mathbb{N}$: the ways of generating an initial solution in ILSQN and swapping two polygons in SWAPTWOPOLYGONS.

  In order to generate an initial solution, ILSQN uses FINDBESTPOSITION if $\sigma = \mathtt{I}$ or $\mathtt{C}$, and uses FINDBESTPOSITION2 if $\sigma \in \mathbb{N}$, where $\sigma \in \mathbb{N}$ is the parameter K of FINDBESTPOSITION2.

  SWAPTWOPOLYGONS swaps the centroids of two given polygon if $\sigma = \mathtt{C}$, swaps them by FINDBESTPOSITION if $\sigma = \mathtt{I}$, and swaps them by FINDBESTPOSITION2 if $\sigma \in \mathbb{N}$, where $\sigma \in \mathbb{N}$ is the parameter K of FINDBESTPOSITION2.

We measure the *efficiency* of a solution by the ratio

$$\frac{\text{the total area of the polygons}}{\text{the area of the container}}.$$

We investigated the effects of the parameters by the following four computational experiments. We choose six benchmark instances (SHAPES0, SHAPES1, SHAPES2, SHIRTS, SWIM, TROUSERS), and conducted 5 runs with the time limit of each run being 10 minutes.

We first examined the effect of π_{side}. Figure 12 shows the average efficiency of five runs; the graph on the left shows the result of $S_{\mathrm{init}} = \mathtt{sort}$, and the one on the right shows the result of $S_{\mathrm{init}} = \mathtt{random}$, where we set $r_{\mathrm{dec}} = 0.04, r_{\mathrm{inc}} = 0.01, N_{\mathrm{mo}} =$

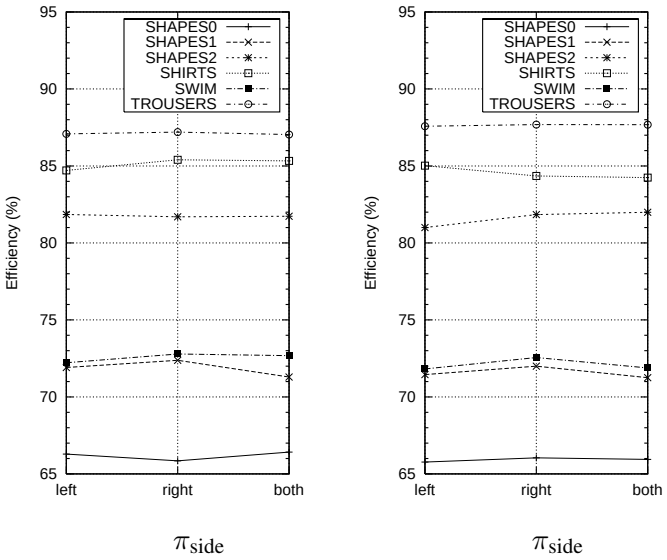

Fig. 12. Average efficiencies against π_{side} (left: $S_{\text{init}} = \texttt{sort}$, right: $S_{\text{init}} = \texttt{random}$)

200, and $\sigma = \texttt{I}$. These results indicate that π_{side} does not have much effect on the efficiency.

We second examined the effect of r_{dec} and r_{inc}. Figure 13 shows the average efficiency of five runs; the graph on the left shows the result of $S_{\text{init}} = \texttt{sort}$, and the one on the right shows the result of $S_{\text{init}} = \texttt{random}$, where we set $N_{\text{mo}} = 200$, $\pi_{\text{side}} = \texttt{right}$, $\sigma = \texttt{I}$. From these results, we observe that the efficiency is almost same for $r_{\text{dec}} \leq 0.06$.

Next, we investigated the effect of N_{mo}. Figure 14 shows the average efficiency of five runs; the graph on the left shows the result of $S_{\text{init}} = \texttt{sort}$, and the one on the right shows the result of $S_{\text{init}} = \texttt{random}$, where we set $r_{\text{dec}} = 0.04, r_{\text{inc}} = 0.01$, $\pi_{\text{side}} = \texttt{right}$, and $\sigma = \texttt{I}$. These results indicate that the efficiency is slightly better for $50 \leq N_{\text{mo}} \leq 300$.

Finally, we checked the effect of σ. Figure 15 shows the average efficiency of five runs; the graph on the left shows the result of $S_{\text{init}} = \texttt{sort}$, and the one on the right shows the result of $S_{\text{init}} = \texttt{random}$, where we set $r_{\text{dec}} = 0.04, r_{\text{inc}} = 0.01$, $\pi_{\text{side}} = \texttt{right}$, and $N_{\text{mo}} = 200$. These results indicate that the efficiency is clearly worse for $\sigma = \texttt{C}$, the efficiency is better for $100 \leq \sigma \leq 800$, and the best result is obtained when $S_{\text{init}} = \texttt{sort}$ and $\sigma = 800$. We can also observe from Figures 12–15 that S_{init} does not have much effect on the efficiency.

Based on these observations, we set $S_{\text{init}} = \texttt{sort}$, $\pi_{\text{side}} = \texttt{right}$, $r_{\text{dec}} = 0.04$, $r_{\text{inc}} = 0.01$, $N_{\text{mo}} = 200$, and $\sigma = 800$ for the computational experiments of all benchmark instances in Section 7.3.

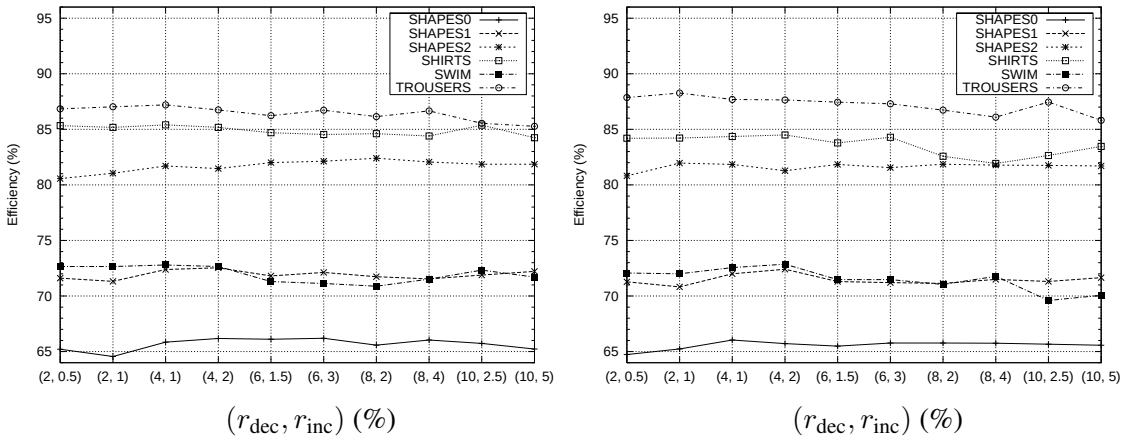

Fig. 13. Average efficiencies against $(r_{\mathrm{dec}}, r_{\mathrm{inc}})$ in % (left: $S_{\mathrm{init}} = \texttt{sort}$, right: $S_{\mathrm{init}} = \texttt{random}$)

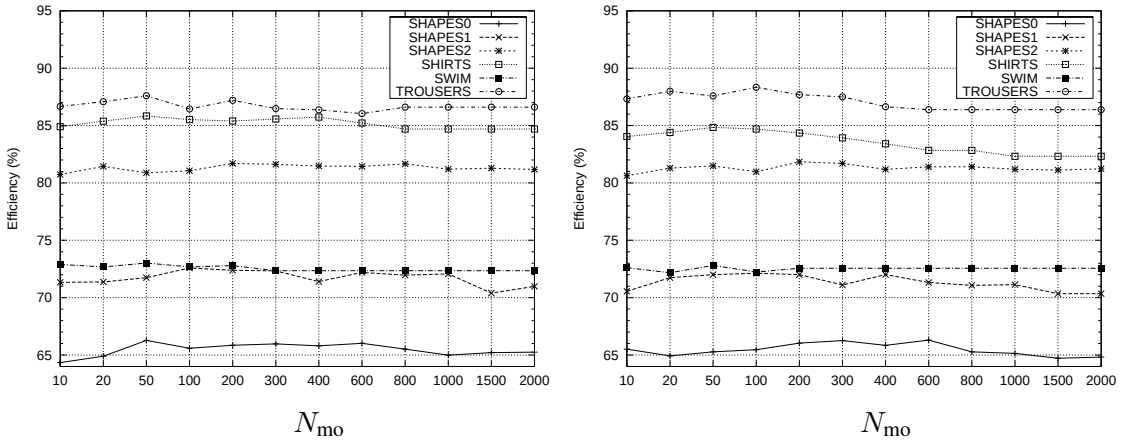

Fig. 14. Average efficiencies against N_{mo} (left: $S_{\mathrm{init}} = \texttt{sort}$, right: $S_{\mathrm{init}} = \texttt{random}$)

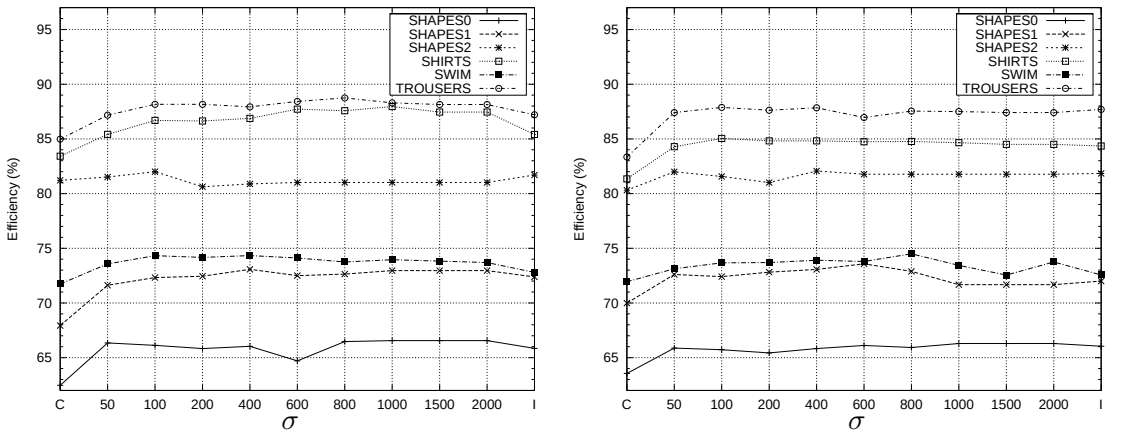

Fig. 15. Average efficiencies against σ (left: $S_{\mathrm{init}} = \texttt{sort}$, right: $S_{\mathrm{init}} = \texttt{random}$)

7.3 Results

In this subsection, we show the computational results of our algorithm ILSQN comparing it with other existing algorithms. We ran algorithm ILSQN ten times for each instances listed in Table 1 and compared our results with those reported by Gomes and Oliveira [8] (denoted as "SAHA"), Burke et al. [9] (denoted as "BLF") and Egeblad et al. [10] (denoted as "2DNest"). Table 2 shows the best and average length and efficiency in % of ILSQN and the best efficiency in % of the other algorithms. The column EF shows the efficiency in %. The best results among these algorithms are written in bold typeface. Table 3 shows the computation time (in seconds) of the algorithms.

Gomes and Oliveira [8] did not use time limit but stopped their algorithm by other criteria. They conducted 20 runs for each instance and the best results of the 20 runs are shown in Table 2, while their computation times in Table 3 are the average computation time of the 20 runs.

Burke et al. [9] tested four variations of their algorithm, and conducted 10 runs for each variation. Their results in Table 2 are the best results of the 40 runs, which are taken from Table 5 in [9]. They limited the number of iterations for each run, and their computation time in Table 3 is the time spent to find the best solution reported in Table 2 in the run that found it (i.e., the time for only one run is reported). Since they conducted experiments for instances ALBANO, DIGHE1 and DIGHE2 with different orientations from the others, we do not include the results.

Egeblad et al. [10] and we conducted experiments using the time limits for each run shown in Table 3.

Although our total computation time of all runs for each instance is not so long compared with SAHA [8] and 2DNest [10], ILSQN obtained the best results for 8 instances out of the 15 instances in efficiency of the resulting layouts and also obtained the results with almost equivalent efficiency to the best results for some instances. It achieved the same efficiency as 2DNest [10] for instance SHAPES1, but the layouts are different. The computation time of BLF [9] is much shorter than that of ILSQN, and ILSQN obtained better results in efficiency than those BLF [9] obtained for all instances. Figure 16 shows the best layouts obtained by ILSQN for all instances.

8 Conclusions

We proposed an iterated local search algorithm for the overlap minimization problem consolidating a separation algorithm based on nonlinear programming and a

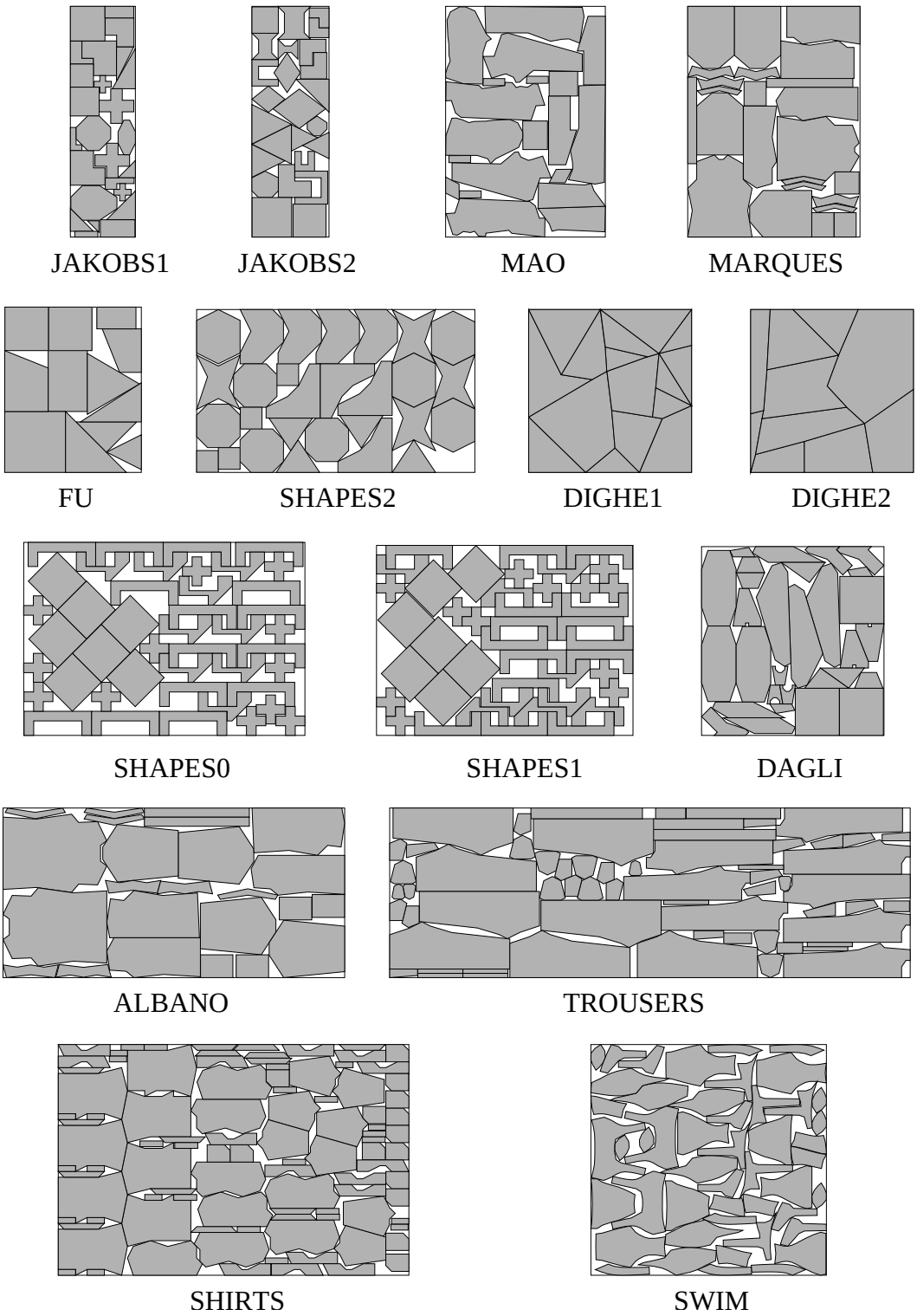

Fig. 16. The best solutions obtained by ILSQN

Table 2
The length of ILSQN and the efficiency in % of the four algorithms

Instance	ILSQN				SAHA [8]	BLF [9]	2DNest [10]
	Average		Best		Best	Best	Best
	Length	EF (%)	Length	EF (%)	EF (%)	EF (%)	EF (%)
ALBANO	9990.23	87.14	9874.48	**88.16**	87.43	–	87.44
DAGLI	59.11	85.80	58.02	**87.40**	87.15	83.7	85.98
DIGHE1	110.69	90.49	100.11	99.89	**100.00**	–	99.86
DIGHE2	120.43	84.21	100.01	99.99	**100.00**	–	99.95
FU	32.56	87.57	31.43	90.67	90.96	86.9	**91.84**
JAKOBS1	11.56	84.78	11.28	86.89	†*78.89	82.6	**89.07**
JAKOBS2	23.98	80.50	23.39	**82.51**	77.28	74.8	80.41
MAO	1813.38	81.31	1766.43	83.44	82.54	79.5	**85.15**
MARQUES	79.72	86.81	77.70	89.03	88.14	86.5	**89.17**
SHAPES0	60.02	66.49	58.30	**68.44**	66.50	60.5	67.09
SHAPES1	54.79	72.83	54.04	**73.84**	71.25	66.5	**73.84**
SHAPES2	26.44	81.72	25.64	**84.25**	83.60	77.7	81.21
SHIRTS	61.28	88.12	60.83	**88.78**	†86.79	84.6	86.33
SWIM	5928.23	74.62	5875.17	**75.29**	74.37	68.4	71.53
TROUSERS	245.58	88.69	242.56	89.79	**89.96**	88.5	89.84

* The value has been corrected from the one reported in [8] according to the information sent from the authors of [8].
† Better results were obtained by a simpler greedy approach (GLSHA)[8]: 81.67% for JAKOBS1 and 86.80% for SHIRTS.

swapping operation of two polygons, and incorporated it into our algorithm ILSQN for the irregular strip packing problem. We showed through computational experiments that ILSQN is competitive with existing algorithms, updating the best known solutions for several benchmark instances.

It is left as future work to combine our algorithm and the multi-stage approach [8] to obtain better solutions quickly. For this approach, we need to develop an algorithm to automatically cluster polygons by some criteria. It is also left to extend our algorithm to handle rotations of any angle or other shapes such as those with curved lines or 3-dimensional objects. Furthermore, computation of a good lower bound of the irregular strip packing problem is also important to evaluate heuristic algorithms or to develop exact algorithms.

Acknowledgements

This research was partially supported by a Scientific Grant in Aid from the Ministry of Education, Science, Sports and Culture of Japan. We would like to thank Toshi-

Table 3
The computation time in seconds of the four algorithms

Instance	*ILSQN Xeon 2.8 GHz 10 runs	†SAHA [8] Pentium4 2.4 GHz 20 runs	‡BLF [9] Pentium4 2.0 GHz 4×10 runs	*2DNest [10] Pentium4 3.0 GHz 20 runs
ALBANO	1200	2257	–	600
DAGLI	1200	5110	188.80	600
DIGHE1	600	83	–	600
DIGHE2	600	22	–	600
FU	600	296	20.78	600
JAKOBS1	600	332	43.49	600
JAKOBS2	600	454	81.41	600
MAO	1200	8245	29.74	600
MARQUES	1200	7507	4.87	600
SHAPES0	1200	3914	21.33	600
SHAPES1	1200	10314	2.19	600
SHAPES2	1200	2136	21.00	600
SHIRTS	1200	10391	58.36	600
SWIM	1200	6937	607.37	600
TROUSERS	1200	8588	756.15	600

* Computation time is the time limit for each run.
† Computation time is the average computation time.
‡ Computation time is the time spent to find the best solution in the run that found it.

hide Ibaraki, Shinji Imahori, Shunji Umetani, Koji Nonobe, and Nobuo Yamashita for numbers of helpful comments and suggestions.

References

[1] B. K. Nielsen, A. Odgaard, Fast neighborhood search for the nesting problem, Technical report 03/03, DIKU, Department of Computer Science, University of Copenhagen (2003).

[2] M. Adamowicz, A. Albano, Nesting two-dimensional shapes in rectangular modules, Computer-Aided Design 8 (1) (1976) 27–33.

[3] A. Albano, G. Sapuppo, Optimal allocation of two-dimensional irregular shapes using heuristic search methods, IEEE Transactions on Systems, Man and Cybernetics SMC-10 (5) (1980) 242–248.

[4] M. A. Gomes, J. F. Oliveira, A 2-exchange heuristic for nesting problems, European Journal of Operational Research 141 (2) (2002) 359–370.

[5] J. F. Oliveira, A. M. Gomes, J. S. Ferreira, TOPOS – a new constructive algorithm for nesting problems, OR Spektrum 22 (2) (2000) 263–284.

[6] Z. Li, V. Milenkovic, Compaction and separation algorithms for non-convex polygons and their applications, European Journal of Operational Research 84 (3) (1995) 539–561.

[7] J. A. Bennell, K. A. Dowsland, Hybridising tabu search with optimisation techniques for irregular stock cutting, Management Science 47 (8) (2001) 1160–1172.

[8] M. A. Gomes, J. F. Oliveira, Solving irregular strip packing problems by hybridising simulated annealing and linear programming, European Journal of Operational Research 171 (3) (2006) 811–829.

[9] E. Burke, R. Hellier, G. Kendall, G. Whitwell, A new bottom-left-fill heuristic algorithm for the two-dimensional irregular packing problem, Operations Research 54 (3) (2006) 587–601.

[10] J. Egeblad, B. K. Nielsen, A. Odgaard, Fast neighborhood search for two- and three-dimensional nesting problems, European Journal of Operational Research 183 (3) (2007) 1249–1266.

[11] E. Hopper, B. C. H. Turton, A review of the application of meta-heuristic algorithms to 2D strip packing problems, Artificial Intelligence Review 16 (4) (2001) 257–300.

[12] M. de Berg, O. Cheong, M. Kreveld, M. Overmars, Computational Geometry: Algorithms and Applications, 3rd Edition, Springer, Heidelberg, 2008.

[13] D. C. Liu, J. Nocedal, On the limited memory BFGS method for large scale optimization, Mathematical Programming 45 (3) (1989) 503–528.

[14] J. A. Bennell, K. A. Dowsland, W. B. Dowsland, The irregular cutting-stock problem – a new procedure for deriving the no-fit polygon, Computers & Operations Research 28 (3) (2001) 271–287.

[15] G. D. Ramkumar, An algorithm to compute the Minkowski sum outer-face of two simple polygons, in: Proceedings of the twelfth annual symposium on computational geometry (SCG '96), ACM Press, New York, 1996, pp. 234–241.

[16] P. K. Agarwal, L. J. Guibas, S. Har-Peled, A. Rabinovitch, M. Sharir, Penetration depth of two convex polytopes in 3D, Nordic Journal of Computing 7 (3) (2000) 227–240.

[17] D. Dobkin, J. Hershberger, D. Kirkpatrick, S. Suri, Computing the intersection-depth of polyhedra, Algorithmica 9 (6) (1993) 518–533.

[18] Y. J. Kim, M. C. Lin, D. Manocha, Incremental penetration depth estimation between convex polytopes using dual-space expansion, IEEE Transaction on Visualization and Computer Graphics 10 (2) (2004) 152–163.

[19] A. Aggarwal, L. J. Guibas, J. Saxe, P. W. Shor, A linear time algorithm for computing the Voronoi diagram of a convex polygon, in: Proceedings of the nineteenth annual ACM conference on theory of computing (STOC '87), ACM Press, New York, 1987, pp. 39–45.

[20] F. Chin, J. Snoeyink, C. A. Wang, Finding the medial axis of a simple polygon in linear time, Discrete and Computational Geometry 21 (3) (1999) 405–420.